CS 201 - Fall 2019

Homework Assignment 3

Due: 23:59, Dec 19, 2019

In this homework, you will implement a book collection system. A book collection has a list of genres (e.g., science fiction). Each genre is denoted with a unique name and a list of books under that genre. Each book is identified by a unique name and a list of authors. In your implementation, you **MUST** use linked lists. This homework will have two parts, whose requirements are explained below.

PART A:

To take the final exam, you MUST submit at least this part and MUST get at least half of its points.

This part is a simplified version of the entire system, where the user just creates the book collection and enters the genres only. So the system does not contain any books (so no authors either). In this system, you must keep the genres in a linked list of the `Genre` objects. Thus, you must implement the `Genre` class first. This class is quite simple for Part A, but you will have to extend it for Part B.

1. Below is the required part of the `Genre` class. The name of the class must be `Genre`. The interface for the class must be written in a file called `SimpleGenre.h` and its implementation must be written in a file called `SimpleGenre.cpp`.
   - The `Genre` class keeps the name of the genre as the only data member. Make sure to implement the get function for this data member since we will use them to test your program.
   - Implement the default constructor, which initializes the `genreName` data member. Implement the getter and setter function for this data member. Additionally, implement your own destructor and copy constructor, and overload the assignment operator. Although you may use the default ones for some of these special functions, you are advised to implement them (some may have no statements) so that it will be easier for you to extend them for Part B.
2. Do not delete or modify any part of the given data members or member functions. You are not allowed to define additional functions and data members to this class for Part A.

```
#ifndef __SIMPLE_GENRE_H
#define __SIMPLE_GENRE_H

#include <string>
using namespace std;

class Genre {
public:
   Genre(const string gname = "");
   ~Genre ();
   Genre (const Genre &genreToCopy);
   void operator=(const Genre &right);
   string getGenreName() const;
   void setGenreName(const string gName);

private:
   string genreName;
};
#endif
```

3. Below is the required part of the `BookCollection` class that you must write in Part A of this assignment. The name of the class must be `BookCollection`. The interface for the class must be

written in a file called `SimpleBookCollection.h` and its implementation must be written in a file called `SimpleBookCollection.cpp`. Do not delete or modify any part of the given data members or member functions. You are not allowed to define additional functions and data members to this class for Part A.

```
#ifndef __SIMPLE_BOOKCOLLECTION_H
#define __SIMPLE_BOOKCOLLECTION_H

#include <string>
using namespace std;
#include "SimpleGenre.h"

class BookCollection{
public:
    BookCollection();
    ~BookCollection();
    BookCollection(const BookCollection& bcToCopy);
    void operator=(const BookCollection& right);
    void displayGenres() const;
    bool addGenre(const string genreName);
    bool removeGenre(const string genreName);
    string getName() const;
    void setName(const string bcName);

private:
    struct GenreNode {
            Genre g;
            GenreNode* next;
    };

    string name;
    int genreCount;
    GenreNode* head;
    GenreNode* findGenre(string genreName);

};
#endif
```

You must keep the genres in a linked-list of `GenreNodes` whose head pointer is `GenreNode *head`. In this class definition, you also see the prototype of a private function called `findGenre`. You may want to implement such an auxiliary function and use it in your add and remove functions (then in some other functions for Part B). This function takes the name of a genre, searches it in the linked list of genres, and returns a pointer to the `GenreNode` that contains that genre if the genre exists in the system. Otherwise, it returns NULL. This auxiliary function may help you write more clear codes. However, if you do not want to use it, just define an empty function (with no statements) in your `SimpleBookCollection.cpp` file.

Things to do:

- Implement the default constructor, which creates an empty book collection system. Also overload the assignment operator and implement the destructor and copy constructors.
- Implement the getter/setter functions for the `name` data member.
- Implement the add and remove genre functions whose details are given below:

  Add a genre: This function adds a genre to the system. The name of the genre is specified as a parameter. In this system, genre names are unique. Thus, if the user attempts to add a genre with an already existing name, then display a warning message, do not add the genre and return false.

Otherwise, if the genre does not exist in the system, add the genre to the system and return true. Note that names are case insensitive (i.e., Fantasy and FANTASY are the same thing).

Remove a genre: This function removes a genre from the system. The name of this genre is specified as a parameter. If the genre with the given name exists in the system, remove it from the system and return true. Otherwise, if there is no genre with the given name, do not perform any action and return false. Display a warning message.

Display all genres: This function should display the names of all genres in the system one per line. If the there are no genres in the system, display `--EMPTY--.`

```
Genre name1
Genre name2
. . .
```

3. To test Part A, write your own main function in a separate file. Do not forget to test your code for different cases such that the system is created and the above-mentioned functions are employed. However, do not submit this file. If any of your submitted files contains the main function, you may lose points.

What to submit for Part A?

You should put your `SimpleGenre.h`, `SimpleGenre.cpp`, `SimpleBookCollection.h` and `SimpleBookCollection.cpp` files into a folder and zip the folder. In this zip file, there should not be any file containing the main function. The name of this zip file should be: PartA_secX_Firstname_Lastname_StudentID.zip where X is your section number. Then follow the steps that will be explained at the end of this document for the submission of Part A.

What to be careful about implementation and submission for Part A?

You have to read "notes about implementation" and "notes about submission" parts that will be given at the end of this document.

PART B:

In this part, you will extend the simple book collection system. For this, first, you are supposed to implement the `Book` and `Author` classes whose interfaces are given below, respectively. Do not delete or modify any part of the given data members or member functions. However, you may define additional functions and data members, if necessary.

Put the `Book` class in a file called `Book.h` and it's implementation in `Book.cpp`, and put the `Author` class in a file called `Author.h` and it's implementation in `Author.cpp`. Implement all the functions given in the header files above. You must keep the authors of a book in a linked-list of `BookNodes` whose head pointer is `AuthorNode* head`. In this class definition of a `Book`, you also see the prototype of a private function called `findAuthor`. You may want to implement such an auxiliary function and use it in your add and remove functions. This function takes the id of an author, searches it in the linked list of authors, and returns a pointer to the `BookNode` that contains that author if the author exists in the system. Otherwise, it returns NULL. This auxiliary function may help you write more clear codes. However, if you do not want to use it, just define an empty function (with no statements) just like the version for genres.

```
#ifndef __AUTHOR_H
#define __AUTHOR_H
#include <string>
using namespace std;

class Author{
public:
    Author();
    Author(const int id, const string name);
    int getID() const;
    void setID(const int id);
    string getName() const;
    void setName(const string id);
private:
    string name;
    int id;
};
#endif
```

```
#ifndef __BOOK_H
#define __BOOK_H

#include <string>
using namespace std;
#include "Author.h"

class Book{
public:
    Book();
    Book(const string name);
    ~ Book();
    Book(const Book& bookToCopy);
    void operator=(const Book& right);
    string getName() const;
    void setName(const string bookName);
    bool addAuthor(const int id, const string name);
    bool removeAuthor (const int id);
    void displayAuthors() const;

private:
    struct AuthorNode {
            Author a;
            AuthorNode* next;
    };
    AuthorNode* head;
    string name;

    AuthorNode* findAuthor(int id);
};
#endif
```

Here is some information about the functions to be implemented in the `Book` class:

Add an author: This function adds an author to the book. The id and name of the author are specified as parameters. In this system, author ids are unique. Thus, if the user attempts to enter an author that exists for that book, display a warning message and do not perform the requested action.

Remove an author: This function removes an author from the book. The id of the author is specified as a parameter. If there is no author with the given id, display a warning message and do not perform the requested action.

Display authors: This function lists all authors added to the book. The output should be in the following format. If the there are no authors of the book, display `--EMPTY--`.

```
Author ID1, Author name1
Author ID2, Author name2
. . .
```

Then, extend the `Genre` class from Part A, such that now it keeps the books of a single genre. These books must be kept in another **LINKED-LIST**. Note that the number of books in a genre is not known in advance. Here, do not forget to implement the constructor, destructor, and copy constructor of this `Genre` class. Also do not forget to overload its assignment operator. Otherwise, you may encounter some unexpected run-time errors. This time, the interface of the `Genre` class must be written in a file called `Genre.h`, and its implementation must be written in a file called `Genre.cpp`.

After extending the `Genre` class, now work on the implementation of the following functionalities that your BookCollection system should support:

1. Add a genre
2. Remove a genre
3. Display all genre
4. Add a book to a genre
5. Remove a book from a genre
6. Add an author to a book in a genre
7. Remove an author from a book in a genre
8. Show detailed information about a particular genre
9. Find the genre (s) and book(s) associated with a specific author id

Add a genre: This function adds a genre to the system. The name of the genre is specified as a parameter. In this function, the book list is not specified; the book(s) (and their author(s)) will be added later. In this system, genre names are unique (case insensitive). Thus, if the user attempts to enter a genre with an already registered name, display a warning message and do not perform the requested action. This function is very similar to what you implemented in Part A. But now, for Part B, you will need to create an empty book list for the genre when you add it to the system.

Remove a genre : This function removes a genre from the system. The name of this genre is specified as a parameter. If there is no genre with the given name, display a warning message and do not perform the requested action. Note that this function also clears the book list of the specified genre as well as author lists for the books. This function is very similar to what you will implement in Part A. But now, for Part B, you will need to remove its book list along with the author lists of the books when you remove the genre from the system.

Display all genres: This function lists all genres already registered in the system. The output should be in the following format. If the there are no genres in the system, display `--EMPTY--`.

```
Genre name1
Genre name2
. . .
```

Add a book to the genre: This function adds a book to the book list of a genre. The genre name for which the book is submitted to and the name of the book are specified as parameters. Note that the author list is initially empty and authors will be added later. In this function, you should take care of the following issues:

- If the genre with the specified name does not exist in the system, display a warning message and do not perform the requested action.
- All book names are unique (case insensitive) within the same genre. Thus, if the user attempts to add a book with an existing name in the same genre, display a warning message and do not perform the requested action. However, different genres can have books with the same name. Note that a book can be entered to multiple genres.

Remove a book from the genre: This function removes a book from the book list of a genre. Note that this operation should also clear the author list of the book. The genre name for which the book is submitted to and the name of the book are given as parameters. If there is no genre with the specified name or if the specified book name is not in the book list of the specified genre, display a warning message and do not perform the requested action.

Add an author to a book in a genre: This function adds an author to a book in a given genre. The genre name for which the book will be entered to, and the name of the book are specified as parameters along with the author name and author id. In this function, you should take care of the following issues:

- If the genre with the specified name does not exist in the system or the given book does not exist in that genre, display a warning message and do not perform the requested action.
- All author ids are unique. Thus, if the user attempts to add an author with an existing id in the same author list, display a warning message and do not perform the requested action. However, different books can have authors with the same id.

Remove an author from a book in a genre: This function removes an author from the author list of the given book in the given genre. The id of the author, the genre name to which the book is entered and the name of the book are given as parameters. If there is no genre with the specified name or if the specified book name is not in the book list of the specified genre or if the author id is not in the author list of the corresponding book, display a warning message and do not perform the requested action.

Show detailed information about a particular genre: This function displays all of the information about a genre whose name is specified as a parameter. The output should be in the following format. If the genre with the specified name does not exist in the system, display `--EMPTY--` after the first line. Authors of a book will be listed following the book name and will start after a tab.

```
Genre name
Book name1
   Author ID1, Author name1
   Author ID2, Author name2
Book name2
   Author ID3, Author name3
   Author ID4, Author name4
   Author ID5, Author name5
```

Find the genre(s) and book(s) associated with a specific author: This function lists all the genres whose book lists contain the specified a book which is authored by the given author ID. Multiple genres can

have multiple books, which can have the same author id. The output should be in the following format. If the specified author does not participate in any book, display `--EMPTY--` after the author ID and name. Found book names follow the genre name and start after a tab.

```
Author ID, Author name
Genre name (for the 1st genre)
   Book name (for the 1st book in 1st genre)
   Book name (for the 2nd book in 1st genre)
Genre name (for the 2nd genre)
   Book name (for the 1st book in 2nd genre)
   Book name (for the 2nd book in 2nd genre)
   Book name (for the 3rd book in 2nd genre)
       . . .
```

Below is the required public part of the `BookCollection` class that you must write in Part B of this assignment. The name of the class must be `BookCollection`. The interface for the class must be written in a file called `BookCollection.h` and its implementation must be written in a file called `BookCollection.cpp`. Your class definition should contain the following member functions and the specified data members. However, this time, if necessary, you may also define additional public and private member functions and data members in your class. You can also define additional classes in your solution. However, you are not allowed to delete any of the given functions or modify the prototype of any of these given functions.

```
#ifndef __BOOKCOLLECTION_H
#define __BOOKCOLLECTION_H
#include <string>
using namespace std;
#include "Genre.h"
class BookCollection {
public:
       BookCollection();
       ~BookCollection();
       BookCollection(const BookCollection& bcToCopy);
       void operator=(const BookCollection& right);
       void addGenre(string genreName);
       void removeGenre (string genreName);
       void displayAllGenres() const;
       void addBook(string genreName, string bookName);
       void removeBook(string genreName, string  bookName);
       void addAuthor(string genreName, string bookName, int authorID, string
authorName);
       void removeAuthor(string genreName, string bookName, int authorID);
       void displayGenre(string genreName);
       void displayAuthor(int authorID);
 private:
       struct GenreNode {
              Genre g;
              GenreNode* next;
       };
       GenreNode *head;
       int genreCount;
       GenreNode* findGenre(string genreName);
};
#endif
```

What to submit for Part B?

You should put your `Book.h, Book.cpp, Author.h, Author.cpp. Genre.h, Genre.cpp, BookCollection.h, and BookCollection.cpp` (and additional .h and .cpp files if you implement additional classes) into a folder and zip the folder. In this zip file, there should not be any file containing the main function. The name of this zip file should be: PartB_secX_Firstname_Lastname_StudentID.zip where X is your section number. Then follow the steps that will be explained at the end of this document for the submission of Part B.

What to be careful about implementation and submission for Part B?

You have to read "notes about implementation" and "notes about submission" parts that will be given just below.

NOTES ABOUT IMPLEMENTATION (for both Part A and Part B):

1. You MUST use **LINKED-LISTs** in your implementation. You will get no points if you use automatically allocated arrays, dynamic arrays or any other data structures such as `vector/array` from the standard library.
2. Do not delete or modify any part of the given data members or member functions for the given classes. You are not allowed to define additional functions or data members for classes in Part A, but you may do so for Part B, if necessary.
3. You ARE NOT ALLOWED to use any global variables or any global functions.
4. Your code must not have any memory leaks for Part A and Part B. You will lose points if you have memory leaks in your program even though the outputs of the operations are correct.
5. Your implementation should consider all names as case insensitive.

NOTES ABOUT SUBMISSION (for both Part A and Part B):

1. Conform to the rules given separately for Part A and Part B. That is, the name of the classes, the name of the .h and .cpp files, and the name of the zip files should conform to the specifications separately given for Part A and Part B. Otherwise, you may lose a considerable amount of points.
2. You need to upload two zip files (one for Part A and the other for Part B) using the upload link on Moodle (all sections) by the deadline. Read "what to submit for Part A" and "what to submit for Part B" sections very carefully.
3. This homework will be graded by your TA, **CAN TAYLAN SARI <can.sari@bilkent.edu.tr>.** You may ask your homework related questions directly to him.
4. No hard copy submission is needed. The standard rules about late homework submissions apply.
5. Do not submit any files containing the main function. We will write our own main function to test your implementations.
6. You are free to write your programs in any environment (you may use either Linux or Windows). Yet, we will test your programs on "`dijkstra.ug.bcc.bilkent.edu.tr`" and we will expect your programs to compile and run on the `dijkstra` machine. If we could not get your program properly work on the `dijkstra` machine, you would lose a considerable amount of points. Therefore, we recommend you to make sure that your program compiles and properly works on "`dijkstra.ug.bcc.bilkent.edu.tr`" before submitting your assignment.